Available online at www.sciencedirect.com

ScienceDirect

IFAC PapersOnline CONFERENCE PAPER ARCHIVE

IFAC-PapersOnLine 49-25 (2016) 384–389

Acceleration of AES Encryption Algorithm Using Field Programmable Gate Arrays

David Smekal [*] **Jakub Frolka** [*] **Jan Hajny** [*]

[*] *Brno University of Technology, The Faculty of Electrical Engineering and Communication, Technicka 12, Brno, Czech Republic (e-mail: smekal@phd.feec.vutbr.cz, hajny@feec.vutbr.cz)*

Abstract: This article deals with encryption on Field Programmable Gate Array (FPGA). The first part describes current state of symmetric and asymmetric cryptography. The following part focuses on the AES algorithm and its implementation in VHDL language. The last part shows testing results of mentioned implementation on card NFB-40G2 containing FPGA from Xilinx series Virtex-7.

Keywords: AES, FPGA, VHDL, implementation, encryption, decryption, NetCOPE

1. INTRODUCTION

Nowadays, great emphasis is placed on security and speed of data transmission. Therefore, recently, when the FPGA (Field Programmable Gate Array) has become more and more accessible, experts began to consider their appropriate use for applications demanding higher performance. FPGAs are programmable gate arrays, which allow us to develop hardware accelerated applications. Their biggest advantage is that they can be programmed after their manufacture which leads to their versatility. Today's generation consists mainly of programmable logic blocks, RAM memory and multiplexers.

One possible way how to take advantage of FPGA can be implementation of asymmetric ciphers. The authors Blum and Paar (1999) implemented a modular exponentiation on FPGA Xilinx XC4000. For asymmetric cipher RSA of length 1024 bits, their design needed 0.75 ms for encryption and 10.18 ms for decryption. The design was later improved and implemented to FPGA Xilinx XC40250XV in Blum and Paar (2001). The improved times for encryption and decryption for RSA cipher of length mentioned above is 0.22 ms for the encryption and 3.1 ms in case of decryption, when the clock frequency was 45.6 MHz. In Daly (2002) Xilinx V1000FG680-6 was used. This design, in case of encryption and decryption using RSA of length 1024 bits could reach data throughput 48.2 kb/s, which corresponds with time of 21.25 ms. In McIvor (2004) architecture for exponentiation was designed, for RSA of length 1024 bits the architecture could reach data throughput 4.79 Mb/s for encryption and 375.54 kb/s for decryption, which corresponds with time 0.214 ms for encryption and 2.73 ms for decryption. These values were obtained from the analysis for FPGA Xilinx XC2V6000. FPGA Xilinx XCV2000E-6 was used in Paar (2005), 1024 bits RSA exponentiation can be performed with frequency 69.4 MHz and with time 6.1 ms.

Other articles deal with the design that is resistant to attacks on side channels. Design in Ciet (2003) was synthesized on Virtex2 XC2V6000 and the testing results for 1024 bit RSA are better than 150 ms. In Fournaris (2010) an algorithm for modular exponentiation, which was tested for 1024 bits on Virtex 5 XC5VLX50T with frequency 274 MHz, was designed.

Another group of experts dealt with the symmetric ciphers. In Elbirt (2001) several algorithms to FPGA were implemented, Rijndael algorithm which forms the basis of AES algorithm was chosen. There are many ways how to implement individual encryption operations to FPGA. Article Arrag (2012) is dedicated to efficient design of hardware implementation of various architectures and the results of their tests are presented. Wiebe (2007) is one of the first articles, which describes hardware implementation of AES-128 algorithm to FPGA (Xilinx Virtex-4 XC4VFX12). The chip works with data with speed up to 640 Mb/s. Data processing speed of one block is around 30 Mb/s.

Section 3 of the article presents our implementation of algorithm AES (Advanced Encryption Standard). AES is considered as a global commercial standard. Our implementation is directed to the platform of network cards, which are based on Virtex-7 Xilinx FPGA.

2. ADVANCED ENCRYPTION STANDARD

Advanced Encryption Standard (AES) NIST (2001), is an algorithm used for data encryption. AES is a part of the symmetric block cipher family, which is working with blocks of data, and they are of the fixed length of 128 bits. These bits are placed to matrix of 4×4, when one cell of matrix corresponds to one byte. One key of length 128, 192 or 256 is used for encryption and decryption. In this paper we are working with 128 bit key. This algorithm can be divided into three parts according to Daemen and Rijmen (1999):

- initial part (Key Expansion, AddRoundKey),
- iteration part – so called round (SubBytes, ShiftRows, MixColumns, AddRoundKey),
- final part (SubBytes, ShiftRows a AddRoundKey).

An expansion of the key is performed at the beginning of the encryption. In the cipher XOR operation between the 128 bit key and the 4×4 state matrix (block of 128 bits of data) is performed. Subsequently, nine iterations which are normally referred to as the round are performed. The number of rounds depends on the length of the key, for 128 bit key it is 9. Every round consists of the substitution of the bytes in the state matrix (SubBytes), rotation of rows (ShiftRows), substitution of columns (MixColumns). The matrix is combined with round's key (AddRoundKey), at the end of each round. The final part consists of the substitution of the bytes, rotation of rows and the last addition of the round key. Bytes of the ciphertext are stored in the resulting matrix. These steps described above are shown in Fig. 1.

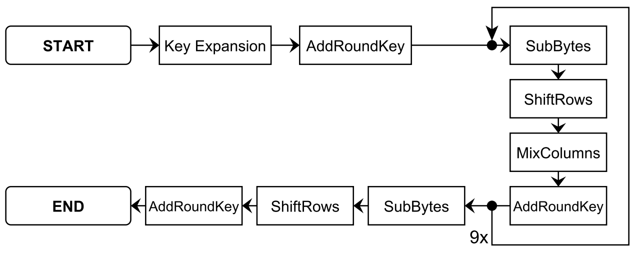

Fig. 1. The steps of the AES algorithm

Decryption is the reverse process of encryption, where the outcome will be a message that was at the beginning of the whole encryption process. The decryption algorithm is performed in reverse order using inverse transformation, because all operations are reversible. The inverse operation is to change the transformation in the reverse order or by using other values. The Inverse transformation for SubBytes transformation is InvSubBytes which utilizes inverse substitution table. InvShiftRows is an inverse transformation for ShiftRows. For MixColumns it is InvMixColumns which utilizes an inverse mixing matrix. The mixing matrix for decryption is shown in Table 3. The last inverse transfomation is InvAddRoundKey, which is only operation XOR.

2.1 Key Expansion

The expansion of the key is performed at the beginning of each encryption. The expansion of the key is used for creation of the keys from the main encryption key to the so-called Round Keys. Obtained Rounds keys are used in transformation AddRoundKey which is applied at each round. If the size of the key is 128 bits, a total of ten sub keys is needed. The original input key is used for initiation part, the newly calculated keys are used for another ten rounds. The number of rounds depends on the length of the key, the exact number is presented in Table 1.

Table 1. The number of rounds

Length of the key	128	192	256
Number of rounds	10	12	14

The key values are represented by matrix 4 × 4 bytes, similarly as the input data of algorithm AES. For better understanding the Key expansion is shown in Fig. 2. This operation carries out the expansion of the key with length of 16 bytes. Values of columns are perceived as array elements, which are labeled W (word). The key has four words, with length of four bytes. The obtained words have index values from 0 to 4. When the number of the iterations is ten, 44 words are needed. Values of the previous key are used for the construction of the new key.

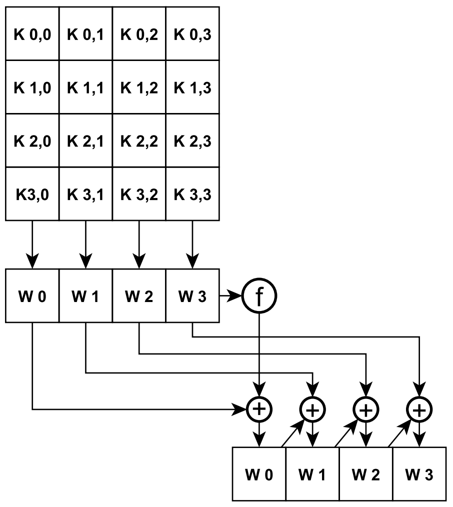

Fig. 2. Key Expansion

The last word is subjected to function f, on which the calculation of other words is based. Operations which are forming function f are shown in Fig. 3. Function f consists of three operations:

- RotWord – function that rotates column upwards by 1 byte,
- SubWord – function that performs substitution SubBytes from substitution table (S-Box),
- XOR with Rcon – operation XOR is used between bytes from columns and Round constant Rcon from Table 2.

Table 2. Rounds constants Rcon

01	02	04	08	10	20	40	80	1b	36
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00

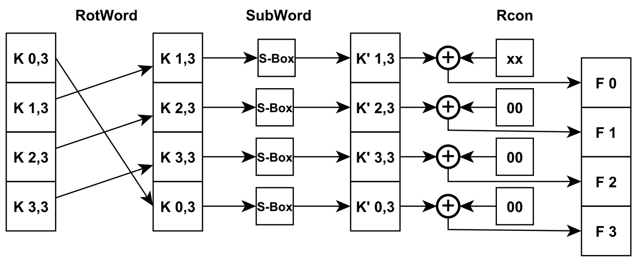

Fig. 3. The function f of the key expansion

After completing these three operations, we have a new word labeled F, which enters further into the process, see Fig. 2. At the end of the operations, we have one new key of length 16 bytes, which is used for one iteration. From